**Final Report of Internship for the "Acertar o Rumo" Program**

**Abstract**

This final report documents the internship conducted at Wisify Tech Solutions under the "Acertar o Rumo" training program. Initially, the project focused on developing a new feature for the Wisify platform—a tool for nutritionists to generate meal plans—which was planned to be integrated with a third-party API from the Fraunhofer Institute. However, during the initial development phase, significant limitations in the external API's flexibility and customization capabilities were identified. This led to a strategic pivot in the project, evolving from a simple integration task into the research and development of a proprietary and intelligent solution.

This document details the complete journey of creating this new system. Using technologies such as Python, Django Rest Framework, and Docker, a sophisticated backend was built, featuring an AI agent that leverages a Retrieval-Augmented Generation (RAG) model combined with a Large Language Model (Llama 3) running locally via Ollama. This approach allows for the generation of highly personalized, varied, and nutritionally balanced meal plans, overcoming the initial challenges. The report covers the system's architecture, the AI-driven generation process, the robust validation and correction mechanisms, and its integration with the React-based frontend. The result is a more powerful, flexible, and strategic feature that adds significant value to the Wisify platform.

**Keywords:** Artificial Intelligence, Meal Plan Generation, Python, Django Rest Framework, Large Language Models (LLM), Retrieval-Augmented Generation (RAG), React, Docker, Health Technology.

**Resumo**

Este relatório final documenta o estágio realizado na Wisify Tech Solutions, no âmbito do curso de formação "Acertar o Rumo". O projeto centrou-se, inicialmente, no desenvolvimento de uma nova funcionalidade para a plataforma Wisify—uma ferramenta para nutricionistas gerarem planos alimentares—cuja implementação estava prevista ser feita através da integração de uma API de terceiros, do Instituto Fraunhofer. Contudo, durante a fase inicial de desenvolvimento, foram identificadas limitações significativas na flexibilidade e capacidade de personalização da API externa. Este diagnóstico levou a uma reorientação estratégica do projeto, que evoluiu de uma tarefa de integração para a investigação e desenvolvimento de uma solução proprietária e inteligente.

Este documento detalha todo o percurso de criação deste novo sistema. Recorrendo a tecnologias como Python, Django Rest Framework e Docker, foi construído um backend sofisticado, que integra um agente de Inteligência Artificial (IA) que utiliza um modelo de Geração Aumentada por Recuperação (RAG) combinado com um *Large Language Model* (Llama 3) a correr localmente através do Ollama. Esta abordagem permite a geração de planos alimentares altamente personalizados, variados e nutricionalmente equilibrados, superando os desafios iniciais. O relatório abrange a arquitetura do sistema, o processo de geração orientado por IA, os robustos mecanismos de validação e correção, e a sua integração com o frontend desenvolvido em React. O resultado é uma funcionalidade mais poderosa, flexível e estratégica, que acrescenta um valor significativo à plataforma Wisify.

**Palavras-chave:** Inteligência Artificial, Geração de Planos Alimentares, Python, Django Rest Framework, Large Language Models (LLM), Geração Aumentada por Recuperação (RAG), React, Docker, Tecnologia da Saúde.

**Introduction**

This report documents the work undertaken during a professional internship at Wisify Tech Solutions, a company at the forefront of health and wellness technology. The internship was a core component of the "Acertar o Rumo" program, designed to bridge academic knowledge with real-world software development practices.

**1.1. Company Profile: Wisify Tech Solutions**

Wisify Tech Solutions is an innovative technology company dedicated to developing advanced solutions for assessment, screening, and prescription in the health, sports, and nutrition sectors. Its mission is to optimize human performance and contribute to the advancement of research and education by combining academic knowledge with technological expertise, delivering solutions that improve people's quality of life. The company distinguishes itself by its ability to transform academic breakthroughs into high-impact technological products with a global reach. A prime example is **Lipowise**, a portable body assessment device that evolved from the university-developed Lipotool into a sophisticated solution integrated with the company's mobile applications.

This environment of practical innovation naturally led the company to identify a critical need within its core platform: a powerful tool for the automatic creation of personalized meal plans. The work developed during this internship was central to this strategic objective. Rather than relying on third-party services with inherent limitations, Wisify made the strategic decision to invest in developing its own proprietary technology. This project, therefore, represents the company's commitment to building highly tailored and intelligent solutions, positioning Wisify as a leader in nutrition technology by offering an integrated platform that empowers health and sports professionals to enhance the well-being of their clients.

This approach, which combines technical and scientific rigor with a keen focus on market needs, results in intuitive, accessible, and highly functional products. This strategy, coupled with its adaptability to new technologies and trends, reinforces Wisify's commitment to being a driving force of innovation in the health and wellness sector.

**1.2. Academic and Internship Context**

This internship was designed to bridge academic theory with real-world application, providing a practical, hands-on experience in a professional software development environment. The initial academic objective was to develop a full-stack application that would be functional across both web and mobile platforms. The plan was to build the frontend interface using React, with the potential to extend to mobile using React Native, while the backend would be powered by a robust API built with Python and the Django Rest Framework. The entire development workflow was to be standardized using Docker to ensure consistency and portability across different systems.

Throughout the initial phase of the internship, the focus was on applying full-stack development principles, creating a practical experience from the initial conception of the application to its implementation and testing. This process was managed using agile methodologies, which fostered a collaborative and efficient approach, allowing for the constant delivery of new features and adaptation to the project's evolving needs. This foundational experience in building a full-stack solution proved to be essential, providing the core competencies in Django, React, and Docker that became critical when the project's direction pivoted towards a more complex, AI-driven challenge. The skills acquired in this first phase laid the groundwork for the successful development of the advanced solution detailed in this report.

**1.3. Initial Project Scope: Third-Party API Integration**

The project began with a clearly defined task: to develop a frontend interface for a new feature in the Wisify platform. This feature would allow nutritionists to automatically generate meal plans for their patients. The backend logic for this generation was to be handled by a third-party API provided by Fraunhofer. My initial responsibilities were to:

1. Develop a React-based frontend to capture patient data (weight, height, dietary restrictions, etc.).
2. Send this data to the Fraunhofer API.
3. Receive the generated meal plan and display it in a clear, intuitive, and user-friendly interface.
4. Develop a backend layer using Django Rest Framework to act as a bridge, manage API keys, and store the generated plans.

**1.4. The Strategic Pivot: The Need for a Custom Solution**

During the initial development and testing phase, significant limitations with the third-party API became apparent. The service, while functional, operated as a "black box," offering minimal room for customization. The structure of the generated meal plans was rigid, and it was impossible to adjust the underlying food databases, nutritional calculations, or generation logic to meet the specific, nuanced needs of Wisify's professional user base.

This critical juncture led to a strategic pivot. Rather than investing further in a restrictive external tool, the decision was made to develop a proprietary meal plan generator from the ground up. This shift transformed the project's scope and complexity, moving it from a relatively straightforward integration task to a challenging and innovative research and development effort. The new goal was to build an intelligent, flexible, and powerful system that would become a core asset for Wisify. This report primarily details the journey of designing, building, and deploying this custom AI-powered solution.

**1.5. Report Structure**

This report is structured into eight chapters. **Chapter 1** details the initial problem analysis and the limitations of the third-party solution. **Chapter 2** outlines the technologies and methodologies used. **Chapter 3** describes the high-level architecture of the new system. **Chapter 4** provides a deep dive into the core AI generator component. **Chapter 5** discusses the frontend implementation. **Chapter 6** presents the results and discusses the challenges faced. **Chapter 7** concludes the report with a summary of achievements and learnings. Finally, **Chapter 8** provides references, followed by appendices with source code and other artifacts.

**Chapter 2: Technologies and Development Methodology**

The success of a complex software project, especially one that undergoes a significant strategic pivot, is heavily reliant on a robust technological foundation and an agile development methodology. This chapter provides a detailed examination of the key technologies and the project management framework that enabled the successful transition from a simple API integration to the development of a sophisticated, AI-driven system.

**2.1. Development Methodology: Scrum**

To manage the project's evolving requirements and ensure consistent progress, the team adopted the Scrum framework. This agile methodology was instrumental in maintaining clarity, collaboration, and adaptability, particularly after the decision to pivot the project's direction. Our implementation of Scrum was characterized by standard ceremonies and artifacts:

* **Sprints:** Work was organized into two-week sprints. This short, iterative cycle allowed for rapid feedback and course correction, ensuring the project remained on track even as its fundamental goals were being redefined. Each sprint concluded with a potentially shippable increment of the product.
* **Sprint Planning:** At the start of each sprint, the team would convene to define the sprint goal. We would select high-priority items from the product backlog and break them down into a concrete sprint backlog, which served as our plan for the upcoming two weeks.
* **Sprint Review & Retrospective:** At the end of each sprint, a review was held to demonstrate the work completed to the team. This was followed by a retrospective, where the team reflected on the past sprint to identify opportunities for process improvement. This continuous feedback loop was crucial for navigating the challenges that arose during the development of the new AI system.
* **Kanban Board:** A digital Kanban board on GitHub was used for task tracking. This provided a transparent, real-time view of the status of all work items (Backlog, Ready, In Progress, In Review, Done), facilitating clear communication and a shared understanding of project progress.

This structured yet flexible approach allowed the team to efficiently manage the shift in project direction, ensuring that the new, more ambitious goals were broken down into manageable tasks and that development remained aligned with the company's strategic objectives.

**2.2. Backend Technologies**

The backend is the engine of the meal plan generator, housing the business logic, database interactions, and the AI agent itself. The technology stack was chosen for its maturity, scalability, and powerful ecosystem.

**2.2.1. Python**

Python was the unequivocal language of choice for the backend. Its clean, readable syntax and extensive standard library promote rapid development and maintainability. However, the primary driver for its selection was its unrivaled position as the de facto language for data science and artificial intelligence. The pivot towards an AI-driven solution made this choice even more critical. Python's vast ecosystem of libraries—such as NumPy and Pandas for numerical and data analysis, and frameworks like LangChain for orchestrating LLM interactions—made it the only practical choice for building a custom, intelligent system.

**2.2.2. Django and Django Rest Framework (DRF)**

Django, a high-level "batteries-included" Python web framework, provided the foundational structure for the entire backend. Django’s core components were instrumental in accelerating development:

* **Object-Relational Mapper (ORM):** The Django ORM abstracted away direct SQL queries, allowing us to define our database models as Python classes and interact with the database using idiomatic Python code. This increased development speed and reduced the likelihood of SQL injection vulnerabilities.
* **Admin Interface:** Django's built-in administrative interface was invaluable for managing and inspecting the recipe and meal plan data during development, providing a ready-made UI for database operations without requiring any frontend work.

Layered on top of Django, the Django Rest Framework (DRF) was used to build the RESTful API that serves as the communication bridge between the frontend and the backend. DRF excels at simplifying the creation of web APIs. Its powerful Serializer classes allowed us to easily convert complex data types, such as our Django model instances, into native Python datatypes that could then be rendered as JSON. This allowed for the rapid development of secure, browsable, and scalable endpoints for creating, retrieving, and modifying meal plans.

**2.3. Frontend Technologies**

**2.3.1. React**

React was used to build the user interface (UI) for the meal planning feature. As a declarative library, React simplifies the process of creating complex UIs by allowing developers to describe what the UI should look like for a given state. Its use of a Virtual DOM provides a significant performance advantage by minimizing direct manipulation of the actual browser DOM. Its component-based architecture was particularly beneficial, allowing for the creation of encapsulated, reusable UI elements (e.g., a MealCard, a DayView). This simplified development and ensured a consistent look and feel across the application. The initial task was to use React to display data from the third-party API. After the pivot, this frontend was seamlessly adapted and expanded to interact with our custom-built DRF backend, providing an intuitive Single-Page Application (SPA) for nutritionists to trigger the AI generation and view the resulting personalized meal plans.

**2.4. Infrastructure and DevOps**

**2.4.1. Docker**

Docker was a cornerstone of the development workflow and played a critical role in ensuring environmental consistency. A key personal challenge of the internship was adapting from a Windows-based development environment to the macOS used at Wisify. Docker completely eliminated this challenge by enabling the creation of lightweight, portable containers.

By defining the entire application environment in a Dockerfile and orchestrating the services (the Django application, the database, and the Ollama LLM service) with docker-compose.yml, we created a one-command setup process for any new developer joining the project. This practice of "containerization" was vital for:

* **Consistency:** Eliminating the classic "it works on my machine" problem by ensuring the application and its dependencies were identical for every team member.
* **Portability:** Simplifying the process of moving the application between development, testing, and production environments with high fidelity.
* **Isolation:** Keeping all dependencies for this project neatly separated from other projects or system-level configurations on the same machine.

**2.5. AI and Machine Learning**

The decision to build a proprietary generator necessitated a foray into the world of applied AI. The tools were chosen to allow for rapid, powerful, and, crucially, private experimentation and deployment.

**2.5.1. Ollama and Llama 3**

For the Large Language Model (LLM) component, we utilized Ollama, an open-source tool designed to simplify the process of running powerful LLMs on local hardware. This was a critical strategic choice that offered several compelling advantages over cloud-based AI services:

* **Privacy and Security:** Meal plan generation can involve sensitive user health data. Processing this data on-premise using a locally-run LLM via Ollama ensured that no private information ever left the company's infrastructure, a paramount concern for any health-tech application.
* **Cost-Effectiveness:** Running the model locally completely eliminated the per-call costs associated with commercial AI APIs. This was particularly beneficial during the development phase, which required thousands of iterative calls to the model for testing and prompt engineering.
* **Control and Customization:** Ollama provides direct access to a variety of cutting-edge open-source models. For this project, we selected Llama 3, a state-of-the-art model from Meta AI, known for its strong reasoning, instruction-following capabilities, and proficiency in generating structured formats like JSON.

The interaction with the Ollama-served Llama 3 model from within our Django application was facilitated by the langchain-ollama library. This library acted as the "glue," providing a seamless Python interface for sending our detailed, retrieval-augmented prompts to the local LLM instance and receiving the generated content back for validation and processing. This setup empowered us to build a sophisticated, private, and cost-effective AI agent directly within our backend infrastructure.

**Chapter 3: Work Performed and System Development**

The development process throughout this internship was not linear but rather an evolution in two distinct phases. It began with a well-defined integration project that, due to technical and strategic challenges, pivoted into a more ambitious and innovative development effort. This chapter details the work performed in both phases, showcasing the project's journey from a simple integration to the creation of a sophisticated AI-powered system.

**3.1. Initial Phase: Integration of the Third-Party API**

The initial objective was to develop a critical feature for the Wisify platform: a tool to automatically generate personalized meal plans for patients. The strategy was to integrate an external API from the Fraunhofer Institute, which would handle the generation logic based on patient data such as weight, height, dietary restrictions, and caloric needs. My work in this phase was divided into two main streams: frontend development for visualizing the meal plans and backend development to integrate the data generated by the API into the company's platform.

**3.1.1. Technical Skills Development**

As Python and the Django Rest Framework (DRF) were new technologies to me, a significant part of the initial phase was dedicated to training. I enrolled in the "Build a Backend REST API with Python & Django - Advanced" course on the Udemy platform. This course was invaluable, providing a solid foundation in Python-based API development. The knowledge acquired was immediately applied to the project, particularly in designing the backend system to communicate with the external API and manage the resulting data. This self-directed learning was essential for completing the internship tasks and contributing effectively to the project's success.

**3.1.2. Frontend Development for API Visualization**

The first practical task was to develop the frontend interface. Using React, I created an intuitive and functional page designed to display the meal plans generated by the Fraunhofer API. This interface clearly presented all the relevant information for each plan, including daily meals, nutritional restrictions, and calorie distribution throughout the week. This frontend was designed to allow nutritionists to easily visualize the data, bridging the gap between the raw API output and a user-friendly experience. Furthermore, this interface was conceived as a foundation for the future integration of the system into Wisify's mobile application, ensuring a fluid and consistent transition between desktop and mobile versions.

**3.1.3. Initial Backend and CRUD System**

Concurrently, the second work stream focused on developing the backend responsible for integrating the meal plans into the Wisify platform. For this, I began developing a CRUD (Create, Read, Update, Delete) system. The purpose of this system was to ensure that after a plan was generated by the external API, its data would be correctly stored in the platform's database and associated with the correct patient. This backend would allow nutritionists to manage and modify the meal plans, making adjustments as needed and ensuring that all changes were reflected in real-time while maintaining data consistency. Using DRF, I started creating the specific API endpoints that would facilitate communication between the frontend and the backend, enabling efficient storage, updating, and retrieval of the data.

**3.2. The Strategic Pivot: From Integration to Innovation**

It was during the development of this initial system that the limitations of the third-party API became apparent. The "black-box" nature of the service, its rigid structure, and the inability to customize the generation logic presented a significant strategic risk. Relying on such an inflexible external tool would limit Wisify's ability to innovate and tailor the product to the specific needs of its professional users.

This realization led to a crucial strategic pivot. Instead of continuing down the path of simple integration, the decision was made to halt the development of the CRUD system for the external API and instead invest in creating a proprietary, intelligent meal plan generator. This transformed the project's scope, elevating it from a standard integration task to a research and development effort focused on building a core intellectual asset for the company.

**3.3. Final Phase: Development of the Proprietary AI Generator**

The second phase of the internship was dedicated entirely to building this new system from the ground up. The foundational skills in Django and React acquired in the first phase were now applied to a much more complex challenge. The simple CRUD system was abandoned, and its backend logic was completely re-architected to house a sophisticated AI agent. This new system, detailed in the subsequent chapters, involved:

* Designing a flexible database schema capable of supporting a wide variety of meal structures and recipes.
* Implementing a deterministic algorithm for reliable, rule-based meal plan generation.
* Engineering a Retrieval-Augmented Generation (RAG) system that combines the power of a Large Language Model (Llama 3) with Wisify's own recipe data to produce creative and varied plans.
* Building robust validation and correction layers to ensure the AI's output always adheres to strict nutritional and structural rules.
* Adapting the React frontend to communicate with the new proprietary API, providing a seamless user experience for the more powerful and flexible backend.

This shift not only resulted in a far superior final product but also provided a much richer and more challenging learning experience, aligning perfectly with the advanced objectives of the internship**.**

**Chapter 4: System Architecture and Design**

A robust and scalable architecture is the foundation upon which any complex software system is built. Following the strategic pivot, significant effort was dedicated to designing an architecture that was not only capable of supporting the immediate needs of an AI-powered meal plan generator but was also flexible enough to accommodate future enhancements. This chapter details the high-level system overview, the intricacies of the database model, the logic behind the dual-generator approach, and the conceptual flow of the user interface.

**4.1. High-Level System Architecture**

The system is designed as a classic three-tier architecture, composed of a frontend client, a backend server, and a database. However, the backend is further subdivided to isolate the intensive processing of the AI agent.

1. Frontend (Client): A React-based Single-Page Application (SPA) that provides the user interface for nutritionists. Its role is to gather user input (target calories, dietary goals) and send a request to the backend API. It is also responsible for receiving the generated meal plan in JSON format and rendering it in a human-readable, interactive format.
2. Backend (Server): A Django application serving a RESTful API via the Django Rest Framework (DRF). This layer handles user authentication, data validation, and standard CRUD operations. When a request for a meal plan is received, it acts as a controller, initiating the generation process.
3. Generator Service: This is the core of the system, encapsulated within a Django management command (create\_personalized\_mealplan\_2.py). It contains the business logic for both the AI and deterministic generators. It communicates with the database to fetch recipes and with the Ollama service to interact with the LLM. It runs as an asynchronous task to avoid blocking the main API server.
4. Database: A PostgreSQL database that stores all persistent data, including users, recipes, ingredients, and the generated meal plans. The structure of this database is defined by the Django ORM models.
5. AI Service (Ollama): An external-but-local service that runs the Llama 3 language model, accessible to the Generator Service via an API call.

The typical flow is as follows: The nutritionist uses the Frontend to request a plan -> The request hits the Backend API -> The API triggers the Generator Service -> The Generator Service fetches data from the Database and sends a prompt to the Ollama AI Service -> The Generator Service validates the AI response and stores the final plan in the Database -> The Backend API notifies the Frontend that the plan is ready to be viewed.

**4.2. Database Model Design**

The database schema, defined in models.py, is the structural blueprint of the application. It was designed to be both normalized and flexible, allowing for complex relationships between different data entities.

**4.2.1. Core Entities: User, Recipe, and Ingredient**

At the heart of the system are the fundamental data models:

* User: This model extends Django's default user model to include profile information crucial for meal plan generation, such as height, weight, date\_of\_birth, and physical\_activity level. It also includes a dietary\_preferences field, linking to the Tag model, which allows the system to filter recipes based on user preferences (e.g., 'vegetarian', 'vegan').
* Ingredient: Represents a single food item (e.g., "Chicken Breast", "Brown Rice"). It contains nutritional information (via one-to-one relationships with In100g, Vitamins, Minerals, etc.), allowing for precise calorie and macronutrient calculations.
* Recipe: Represents a specific dish (e.g., "Grilled Chicken Salad"). A recipe is composed of multiple ingredients through the RecipeIngredient linking model, which specifies the quantity of each ingredient. The Recipe model calculates its total nutritional value (calories, protein, etc.) by aggregating the data from its constituent ingredients. It also has a many-to-many relationship with the Tag model, which is critical for filtering and categorization (e.g., tagging a recipe as 'lunch', 'dinner', 'post-workout').

**4.2.2. Hierarchical Plan Structure: MealPlan, Day, and Meal**

A meal plan is not a flat list of recipes; it has a clear hierarchical structure, which is reflected in the database models:

* MealPlan: This is the top-level container for an entire plan. It is linked to a User and has a title and description.
* MealPlanDay: A MealPlan is composed of multiple MealPlanDay objects. Each day has a specific day\_type (e.g., 'regular', 'workout', 'rest') to which different calorie targets can be applied.
* Meal: Each MealPlanDay consists of several Meal objects. A meal is defined by its meal\_type (e.g., 'breakfast', 'lunch', 'dinner').

This structure (MealPlan -> MealPlanDay -> Meal) allows for a highly organized and easily queryable representation of the entire plan.

**4.2.3. Meal Composition: MealPart and the Linking Model**

To provide even finer-grained control, a Meal is not directly linked to recipes. Instead, it is broken down into MealPart objects.

* MealPart: This model defines the constituent parts of a meal, such as 'main course', 'soup', or 'fruit'. A MealPart is defined by its name and the meal\_type it belongs to (e.g., the 'soup' part belongs to the 'lunch' meal type).
* MealPartRecipe: This is the crucial linking model that brings everything together. It connects a specific Recipe to a specific MealPart within a specific Meal. This allows the system to construct a 'lunch' Meal that contains a MealPartRecipe linking a "Chicken Soup" Recipe to the 'soup' MealPart, and another MealPartRecipe linking a "Tuna Salad" Recipe to the 'main course' MealPart.

**4.2.4. Feedback Loop: The UserRecipeFeedback Model**

To enable the system to learn and improve over time, the UserRecipeFeedback model was introduced. This model captures user-specific interactions with recipes, storing information like rating, liked, cooked\_count, and skip\_count. This data is used by the deterministic generator's scoring algorithm to favor recipes that the user has previously enjoyed and avoid those they have disliked, creating a personalized feedback loop.

**4.3. The Dual-Approach Generator Architecture**

A core architectural decision was to not rely solely on the AI. This led to a dual-generator system that combines the creativity of an AI with the reliability of a deterministic algorithm.

**4.3.1. Primary Path: The AI Agent (RAG)**

The default generation path uses the AI agent. This approach, detailed in the next chapter, leverages the Llama 3 model to act as an intelligent scheduler. It is provided with a curated list of candidate recipes (the "Retrieval" part of RAG) and a set of strict rules. Its job is to assemble these building blocks into a creative, varied, and sensible meal plan. This path optimizes for plan quality and user engagement**.**

**4.3.2. Safety Net: The Deterministic Fallback**

Recognizing that AI can sometimes fail (e.g., produce invalid output or the service may be down), a complete, independent deterministic generator was also built. This system uses a mathematical scoring algorithm (score\_recipe) to evaluate and select the best possible recipe for each meal part based on a weighted combination of factors (calorie proximity, tag matching, user feedback). This path is 100% reliable and predictable. If the AI agent fails for any reason, the system automatically falls back to this generator, ensuring that the user always receives a valid meal plan. This path optimizes for reliability and robustness**.**

**4.4. Conceptual Frontend Interaction Flow**

The design of the system architecture directly informs how the frontend should be structured for an optimal user experience. The conceptual flow is as follows:

* Configuration: The nutritionist is presented with a simple form to input the patient's target daily calories, nutritional goal (e.g., 'weight\_loss'), and any key dietary preferences (which translate to tags).
* Initiation: Upon submission, the frontend makes an asynchronous API call to the backend to start the generation process. It does not wait for the result but instead shows a loading or "generating" state to the user, as the process can take several seconds.
* Polling/Notification: The frontend periodically polls the backend (or listens for a real-time notification, e.g., via WebSockets) to check if the plan is ready.
* Display: Once the backend confirms the plan has been generated and stored, the frontend makes another API call to fetch the final JSON data. It then parses this data and renders it in a clear, hierarchical view: days are displayed as tabs or sections, each containing cards for meals, which in turn list the selected recipes for each part.
* Interaction: The interface allows the nutritionist to view details for each recipe, and potentially modify the plan by replacing a recipe, triggering a smaller, targeted backend process to find a suitable substitute**.**

**5.1. Core Component:** create\_personalized\_mealplan\_2.py

This script orchestrates the entire meal plan creation process. It is responsible for fetching user data, querying the recipe database, interacting with the Large Language Model (LLM), and validating and storing the final output.

**5.2. The RAG (Retrieval-Augmented Generation) Strategy**

Instead of relying solely on the LLM's pre-existing knowledge, which can be generic or outdated, we implemented a RAG strategy. This approach grounds the AI's "creativity" in real, validated data from our own database. The process involves:

1. **Retrieval:** Before generating the plan, we retrieve a curated list of candidate recipes from the Wisify database that are relevant to the user's preferences and caloric needs.
2. **Augmentation:** This list of real, available recipes is "augmented" into the prompt we send to the LLM.
3. **Generation:** The LLM uses this context-rich prompt to generate a meal plan, acting as an intelligent "scheduler" that assembles a valid plan from the provided building blocks.

This method significantly improves the factual accuracy and relevance of the output, ensuring the generated plans consist of actual recipes available within the Wisify ecosystem.

**5.4. Step 2: Building the LLM Prompt**

The quality of the AI's output is directly proportional to the quality of the prompt. A significant portion of the development effort was focused on engineering a detailed, robust prompt. The prompt includes:

* A clear definition of the AI's persona ("You are an expert meal planning assistant").
* Strict output requirements (JSON format, required fields, data types).
* The user's specific goals and caloric targets.
* The detailed rules for meal structure (e.g., "Breakfast: 'main course' (required), 'fruit' (optional)...").
* The curated list of candidate recipes for each specific meal part, retrieved in the previous step.

**5.6. Step 4: Validation of the AI-Generated Plan**

LLMs can sometimes fail to follow instructions perfectly. To ensure the integrity of the meal plans, a strict validation function (validate\_ai\_meal\_plan) was implemented. This function programmatically checks the AI's JSON output against a set of critical business rules:

* Does the plan contain the required three day types (regular, workout, rest)?
* Are all required meals present for each day?
* Are all required meal parts present for each meal?
* Are the selected\_recipe\_id values valid and do they exist in the database?
* Do the selected recipes have the correct tags for the meal and part they've been assigned to?
* Is the total daily calorie count within an acceptable tolerance (e.g., ±15%) of the target?

**5.7. Step 5: The Correction and Fixing Mechanism**

If the validation step fails, we don't immediately give up on the AI's output. Instead, we trigger a fix\_ai\_meal\_plan function. This function takes the flawed AI plan and systematically corrects it. It iterates through the plan's structure, enforces the rules, and replaces invalid or missing recipe selections by making deterministic choices using the same select\_recipe\_for\_part logic as the fallback system. This hybrid approach leverages the AI's ability to create a plausible high-level structure, while using deterministic code to guarantee the final details are correct and valid.

This allows us to salvage creative but imperfect outputs from the LLM, increasing the success rate of the AI generation process.

**Chapter 6: Results, Discussion, and Challenges**

This chapter evaluates the outcomes of the project, providing a critical analysis of the final system. It presents a detailed comparison of the two generation methodologies developed, discusses the system's performance, and reflects on the significant challenges overcome during the development lifecycle.

**6.1. Final Product Demonstration**

The final product delivered is a robust backend service, accessible via a secure Django Rest Framework API. The service exposes an endpoint that, given a set of user parameters (e.g., target daily calories, nutritional goal), generates a complete 3-day meal plan. This plan is highly structured, detailing meals for each day (Breakfast, Lunch, Dinner, etc.), breaking down meals into their component parts (e.g., Main Course, Soup), and assigning a specific, existing recipe to each part.

The system is architected as a dual-approach generator:

1. **Primary AI Agent:** By default, it attempts to generate the plan using the RAG-based AI agent.
2. **Deterministic Fallback:** If the AI agent fails for any reason, or if explicitly requested, the system seamlessly falls back to a purely deterministic algorithm, guaranteeing a valid plan is always produced.

The output is a well-formed JSON object, designed for easy consumption by the Wisify frontend, allowing for a rich and interactive presentation to the end-user (the nutritionist).

**6.2. Comparison: AI vs. Deterministic Output**

The decision to build a dual-system generator was a direct consequence of the trade-offs between a purely creative AI approach and a rigidly reliable deterministic one. The two systems offer complementary strengths and weaknesses.

|  |  |  |
| --- | --- | --- |
| **Feature** | **AI-Powered Generator (RAG)** | **Deterministic Generator** |
| **Creativity & Variety** | **Excellent.** The LLM excels at creating novel, human-like combinations of meals, reducing repetition and increasing patient engagement. It can infer subtle relationships between foods that are difficult to encode in rules. | **Limited.** Relies on a scoring algorithm. While effective, it can become repetitive over time, consistently favoring the same set of high-scoring recipes. |
| **Predictability & Reliability** | **Moderate.** The non-deterministic nature of LLMs means outputs can vary. Failures to generate a valid plan are possible, though mitigated by the validation and fixing mechanisms. | **Excellent.** 100% predictable and reliable. For a given set of inputs, it will always produce a valid plan that adheres to all coded rules. It serves as the system's safety net. |
| **Adherence to Constraints** | **Good (with validation).** The AI attempts to follow constraints from the prompt. However, it can make errors, necessitating the validate\_ai\_meal\_plan and fix\_ai\_meal\_plan functions to enforce rules programmatically. | **Perfect.** The system is *built from* the constraints. Its logic is a direct implementation of the business rules, making violations impossible. |
| **Development & Maintenance** | **Complex.** Initial development is faster for high-level structure, but debugging involves "prompt engineering," which is iterative and less precise. Maintenance requires keeping up with evolving AI models and techniques. | **Moderate.** The initial logic is complex to write and test thoroughly. However, debugging is straightforward, as the code execution path is traceable and predictable. |
| **End-User Experience** | **More Engaging.** The variety and "surprise" factor can lead to more interesting and sustainable meal plans for patients. This novelty is a significant feature. | **More Consistent.** Provides a fast, reliable, and "good enough" plan every time. This consistency can be preferable for nutritionists who need a quick and dependable result. |

In essence, the AI generator optimizes for **quality and engagement**, while the deterministic generator optimizes for **reliability and speed**. The final architecture leverages the best of both worlds, using the AI as the primary, innovative engine and the deterministic system as a crucial and robust fallback mechanism.

**6.3. Performance Analysis**

There is a clear performance trade-off between the two approaches.

* **Deterministic Generator:** Execution is extremely fast, typically completing in **under one second**. The process involves direct database queries and in-memory calculations, which are highly optimized.
* **AI Generator:** This process is significantly slower. The primary bottleneck is the LLM inference time. Running the Llama 3 model locally, generating a complete 3-day plan can take anywhere from **5 to 15 seconds**, depending on the complexity of the prompt and the current system load. This duration includes prompt construction, LLM processing, and the final validation/parsing steps.

While the AI's generation time is acceptable for a feature that is not used in real-time, it highlights the importance of the deterministic fallback for scenarios where speed is a critical factor.

**6.4. Key Challenges Encountered**

The project's evolution presented several significant technical and conceptual challenges:

* **LLM Unpredictability and Constraint Adherence:** The single greatest challenge was forcing a creative, generative model to adhere to a rigid set of business rules and a strict JSON schema. The LLM would occasionally "hallucinate" or fail to follow instructions perfectly. This led to the development of the validation-and-correction loop, which became a non-negotiable cornerstone of the system's architecture to ensure data integrity.
* **Iterative Prompt Engineering:** Crafting the perfect prompt was more art than science. It required dozens of iterations to find the right balance between giving the LLM enough context and creative freedom while being prescriptive enough to guarantee a usable output. Small changes in wording could lead to vastly different results.
* **Adapting to a New Technology Stack:** The personal challenge of transitioning from a Windows to a macOS development environment, while simultaneously learning the intricacies of Docker and Django, presented a steep initial learning curve. However, embracing containerization with Docker proved to be the key to managing this complexity and creating a standardized, reproducible environment.

**6.5. Project Management and Team Collaboration**

The Scrum methodology was vital in successfully navigating the project's pivot. When the limitations of the third-party API were confirmed, Scrum provided the framework to formally re-evaluate the project's goals. We were able to effectively discard the old backlog items, engage in a new discovery and planning phase, and populate our backlog with a new set of user stories and tasks that defined the AI generator. This prevented scope creep and ensured the team remained aligned and focused on a new, clear objective, executing on this vision in a structured, iterative manner.

**Chapter 7: Conclusion**

This final chapter summarizes the project's achievements, reflects on the fulfillment of the internship's objectives, discusses the personal and professional growth experienced, and proposes potential avenues for future work.

**7.1. Summary of Achievements**

This internship culminated in the successful design, development, and deployment of a feature that significantly surpassed the original project scope. What began as a frontend integration task evolved into the creation of a proprietary, sophisticated, dual-system meal plan generator. The primary achievement is a production-ready backend service featuring an innovative AI agent. This agent leverages a locally-run Large Language Model within a Retrieval-Augmented Generation (RAG) architecture to create high-quality, varied, and personalized meal plans. The system's robustness is guaranteed by a fully independent, deterministic fallback generator. This completed feature is a significant technical asset for Wisify, providing a powerful and flexible tool for nutrition professionals and a distinct competitive advantage.

**7.2. Fulfillment of Internship Objectives**

The project provided the ideal context to meet and exceed the core objectives outlined for the internship period.

* **Demonstrate Autonomous Performance in Programming Tasks:** The strategic pivot from an integration task to a full-blown development project required a high degree of autonomy. I took ownership of the entire backend lifecycle: from the initial research into AI methodologies and database design to the implementation and testing of the final, complex system. This went far beyond the original objective of simply integrating an external service.
* **Adaptation to New Contexts and Technologies:** A key personal goal was to adapt to Wisify's macOS-based environment. This was successfully achieved, but the project's evolution demanded adaptation on a much larger scale. Mastering a new professional tech stack including Docker, Django, Django Rest Framework, and the entire applied AI toolkit (Ollama, LangChain, prompt engineering) was a core part of the internship's success.
* **Effective Integration and Teamwork:** While much of the backend development was an autonomous task, constant communication and alignment with my company supervisor, Tiago Andrade, and the needs of the frontend team were paramount. This ensured that the API endpoints were well-designed and that the final feature integrated smoothly into the broader Wisify platform, demonstrating effective collaboration in a professional agile environment.

**7.3. Personal and Professional Development**

This internship has been a transformative professional experience. The technical skills acquired in Python, Django, and particularly in the nascent and rapidly evolving field of applied LLM development, are invaluable. The process of building a system with real-world constraints has solidified theoretical knowledge into practical ability.

However, the most significant growth has been in the non-technical domain. The experience of analyzing a failing strategy (reliance on the third-party API), articulating its shortcomings, and then architecting and executing a new, superior solution has been incredibly empowering. It has fostered a robust engineering discipline and a proactive, problem-solving mindset. I conclude this internship not merely with a stronger technical skillset, but with the confidence and experience of having navigated a complex project from conception to completion, transitioning from a student applying known solutions to an engineer creating new ones.

**7.4. Future Work and Potential Enhancements**

The current system provides a powerful foundation, but there are numerous avenues for future enhancement that could further increase its value:

* **Enhanced Personalization and Feedback Loop:** The system could be extended to incorporate more granular user data, such as specific food allergies, ingredient-level dislikes, and preferred cooking complexity. Furthermore, a feedback mechanism could be implemented where nutritionists' modifications to a generated plan are used to fine-tune the recipe scoring algorithm or even provide data for fine-tuning the LLM, creating a system that learns and improves over time.
* **Model and Performance Optimization:** As new and more efficient open-source LLMs become available, the system can be easily updated to leverage them. For wider, production-level deployment, the Ollama service could be moved to a dedicated, GPU-powered server to handle a higher volume of concurrent requests and reduce inference time.
* **Expansion of Generation Logic:** The generator could be enhanced to create plans longer than three days, or to incorporate more complex nutritional constraints, such as macronutrient cycling or specific micronutrient targets.
* **Interactive Frontend:** The frontend could be made more interactive, allowing nutritionists to "lock" certain meals or recipes they like and have the AI re-generate the rest of the plan around those fixed points.

**Chapter 8: Conclusion**

This final chapter summarizes the project's achievements, reflects on the fulfillment of the internship's objectives, discusses the personal and professional growth experienced, and proposes potential avenues for future work.

**8.1. Summary of Achievements**

This internship culminated in the successful design, development, and deployment of a feature that significantly surpassed the original project scope. The project successfully navigated a critical strategic pivot, transitioning from a straightforward integration of a third-party API into the creation of a proprietary, complex AI development project. The final deliverable is a fully functional and intelligent meal plan generation system, a significant technical asset for Wisify.

The system's dual-architecture approach is a key achievement. It delivers personalized, varied, and nutritionally-aware meal plans via its primary AI agent, while guaranteeing reliability and 100% uptime through its deterministic fallback mechanism. This provides a powerful, flexible, and robust tool for nutrition professionals. The entire backend was delivered as a containerized service, complete with a well-documented API, ready for seamless integration into the Wisify platform and demonstrating a complete, end-to-end development cycle.

**8.2. Fulfillment of Internship Objectives**

The project provided the ideal context to meet and exceed the core objectives outlined for the internship period. The evolution of the project allowed for a much deeper and more comprehensive fulfillment of these goals than originally anticipated.

* Full-Stack Development: The project required hands-on, in-depth work across the entire stack. This included designing and implementing a complex database schema with the Django ORM, building a secure and scalable API with Django Rest Framework, architecting the core business logic in Python, and developing a responsive React frontend to consume the final API.
* Proficiency in Modern Technologies: Beyond simple use, the internship fostered true proficiency in essential modern tools. Docker became second nature for creating reproducible development environments, crucial for overcoming the initial macOS adaptation. The agile Scrum methodology was not just followed but was instrumental in managing the project's pivot, providing a real-world lesson in its value for adapting to change.
* Advanced Problem-Solving: The core of the project was an exercise in advanced problem-solving. It involved critically analyzing the limitations of an existing third-party solution, articulating the business case for a new approach, and then architecting a superior, custom alternative from the ground up. The design of the validation-and-correction loop for the AI's output is a prime example of building a robust solution to a complex, non-deterministic problem.
* Cutting-Edge AI Integration: The project went far beyond a simple API call. A key achievement was implementing a state-of-the-art AI system using a local LLM (Llama 3) within a Retrieval-Augmented Generation (RAG) framework. This provided invaluable, hands-on experience in prompt engineering, model interaction, and managing the inherent unpredictability of generative AI, skills that are at the forefront of the technology industry.

**8.3. Personal and Professional Development**

This internship has been a period of immense and accelerated growth, both technically and professionally.

On a technical level, the learning curve was steep and rewarding. I moved beyond academic knowledge to gain professional-level competence in Python and Django, mastering concepts like the ORM, database migrations, and API design with DRF. Adapting to the macOS environment and embracing Docker for containerization has made me a more versatile and efficient developer. The deepest technical growth, however, came from the AI work. I developed practical skills in prompt engineering, implementing the RAG pattern, and building programmatic safeguards for LLM interactions—experience that is both highly relevant and not typically covered in standard curricula.

On a professional level, this experience was transformative. The responsibility of taking ownership of a failing strategy and leading the development of its replacement fostered a strong sense of autonomy and engineering discipline. It taught me the importance of thinking critically about technical decisions not just in terms of implementation, but also in terms of business strategy and long-term value. Interacting with my supervisor and the team honed my communication skills, particularly in explaining complex technical concepts and aligning on development goals. I conclude this internship having transitioned from a student who applies known solutions to an engineer who analyzes problems, architects new solutions, and delivers robust, well-documented products.

**8.4. Future Work and Potential Enhancements**

The current system provides a powerful and stable foundation, but there are numerous exciting avenues for future enhancement that could further increase its value and sophistication:

* Closed-Loop Feedback and Model Fine-Tuning: The UserRecipeFeedback model currently informs the deterministic scorer. The next logical step is to use this data to create a true learning system. The collected data on which recipes are liked, skipped, or modified by nutritionists could be used to periodically fine-tune the open-source LLM, teaching it the specific preferences of the user base and improving its "out-of-the-box" generation quality.
* Advanced Nutritional Constraints: The current system focuses primarily on calories. It could be expanded to handle more complex nutritional goals, such as macronutrient cycling (different protein/carb/fat ratios on different days), or targeting specific micronutrient minimums (e.g., ensuring sufficient iron or Vitamin C over the week).
* Interactive and Dynamic Generation: The frontend could be made more interactive. A feature could be added to allow a nutritionist to "lock" a recipe they like in the generated plan and then click a "regenerate" button. This would trigger a backend process to create a new plan for the remaining slots, keeping the locked recipe in place. This would offer a powerful blend of AI generation and professional human oversight.
* Performance and Scalability Optimization: For wider deployment, the Ollama service could be moved to a dedicated, GPU-powered cloud or on-premise server to handle a higher volume of concurrent requests and significantly reduce inference time. Experimenting with smaller, quantized versions of LLMs could also provide a better balance between performance and generation quality**.**